SISTEMUL DE TIPURI AL LIMBAJULUI C

1. Tipuri predefinite

- *char* – un octet înmagazinând un caracter din mulțimea locală de caractere
- *int* – mulțimea numerelor întregi de pe maşina gazdă
- *float* – nr. în virgulă mobilă, în simplă precizie
- *double* – nr. în virgulă mobilă, în dublă precizie

Tipului *int* i se pot aplica modificatorii **short** şi **long**:

- *short int* – de obicei pe 16 biți
- *long int* – pe minim 32 biți

Lungimile efective pot fi alese la implementare cu următoarea condiție:

Lungime (short) ≥ 16 biți

Lungime (short) ≤ Lungime (int) ≤ Lungime (long)

Modificatorii **signed** şi **unsigned**, introduşi de standardul ANSI al limbajului, se pot aplica tipului *char* şi tipurilor întregi.

Exemple: *unsigned char* – valori între *0* şi *255*

signed char – valori între -*128* şi *127*

(caracterele care pot fi tipărite sunt întotdeauna pozitive)

Deoarece dimensiunile obiectelor întregi şi reale depind de implementare, s-au inclus în fişierele antet standard <limits.h> şi <float.h> constante simbolice pentru toate aceste dimensiuni, împreună cu alte proprietăți legate de maşină şi compilator.

2. Constante de tip enumerare

```
enum boolean  { NU, DA } ;
enum zile  { LUNI=1, MARTI, MIERCURI, JOI, VINERI,
             SAMBATA, DUMINICA } ;
```

- ✓ *Observație:* Nu este obligatoriu să se verifice dacă valorile unor variabile de tip enumerare corepund valorilor specificate pentru tip.

3. Tipuri de date structurate

a) Tabloul

- *Forma generală* a unei declarații de tablou:

  tip-element tablou[expresie-constanta]

  expresie-constanta : dimensiunea tabloului > 0

Exemplu: *int v[10]* definește un vector de 10 întregi. Indicii încep de la zero ⇒ primul element al vectorului este *v[0]* şi ultimul este *v[9]*.

Tablourile pot fi inițializate la declarare:

int x[ ] = { 1, 2, 3 };

Dimensiunea tabloului trebuie cunoscută la compilare ⇒ tablouri statice

- *Tabloul multidimensional* este un tablou de tablouri:
  - *int mat [10] [10]* reprezintă o matrice cu 10 linii şi 10 coloane
  - corepunzător, elementul de indici (*i, j*) va fi selectat astfel: *mat [i] [j]* şi **nu** *mat[i,j]* ca în multe alte limbaje

- *Parametrii formali tablouri* pot fi declarați incomplet: fără precizarea primei dimensiuni:

  int f (char l[ ], int m[ ] [10]) ;

Dimensiunile efective ale tablourilor se vor preciza în momentul apelului funcției, fiind dimensiunile argumentelor.

⇒ se pot descrie funcții cu un grad de generalitate ceva mai mare decât în Pascal, unde era obligatoriu ca dimensiunile parametrului formal şi cele ale argumentului să fie identice.

b) Structura

- Implementează în C produsele carteziene:

```
struct punct {
    int x;
    int y;
} ;
```

Structurile pot fi copiate prin atribuire şi pot fi inițializate la declarare.

struct punct origine = { 0, 0 } ;

Selectarea *c*âmpurilor se face în forma consacrată:

struct punct p;

p.x sau *p.y*

- Spre deosebire de Pascal, în C funcțiile pot returna structuri:

struct punct f (int x, int y)

{ }

- Structurile pot fi îmbricate:

struct dreptunghi {

struct punct p1 ;

struct punct p2 ;

}

În acest caz selectarea câmpurilor se face pe rând:

struct dreptunghi d ;

d. p1. x

c) Uniunea

Implementează în C reuniunile variabilelor:

```
union {
    int i ;
    float f ;
    char c ;
} u ;
```

u - poate reprezenta, după caz, un întreg, un real sau un caracter.

Selecția variantei – similară cu selecția câmpurilor unei structuri: *u. i, u. f, u. c*

- Se permit declarații îmbricate de uniuni, structuri şi tablouri, în orice ordine.
- În ceea ce priveşte reprezentarea în memorie, toate variantele au deplasament zero față de adresa de început ⇒ la un moment dat se poate înmagazina o singură variantă.

Nu se fac verificări privind corespondența între varianta referită şi cea înmagazinată: toată responsabilitatea cade în sarcina programatorului.

- Operațiile permise sunt cele de la structuri.

O uniune nu poate fi inițializată decât cu o valoare de tipul primei variante: întreagă, în cazul lui *u*.

4. Pointeri

- Ca şi în Pascal, declararea unui pointer presupune precizarea tipului obiectelor referite:

```
int x = 1,  y ;
int *p ;  /*  este un pointer la un intreg */
```

Excepție: `void *p1;  /*` poate stoca orice tip pointer `*/`

- În C pointerii pot înmagazina adresele unor obiecte:

```
p = &x ;
```

Accesul la obiectul indicat de pointer – *operația de dereferențiere*:

```
y = *p ;  /*  y ia valoarea 1 */
*p = 0 ;  /*  x ia valoarea 0 */
```

- Se pot crea sinonime, cu consecințele cunoscute.

- **Pointerii permit accesul direct la locația de memorie a unui argument:**

Exemple*:*

```
void schimba1(int x, int y) /* gresit */
    {
        int aux ;
        aux = x;  x = y;   y = aux ;
    }
```

schimba1(a,b) - schimbă doar copiile lui *a* și *b*

```
void schimba2(int *x, int * y)
    {
        int aux ;
        aux = *x;  *x = *y;   *y = aux ;
    }
```

schimba2(&a, &b) /* apel corect */

➢ **Pointerii pot fi utilizați în C şi în relație cu tablourile:**

int a[10];

*int * pa;*

---

pa = &a[0]; /* pa contine adresa lui a[0] */

♦ Dar valoarea unei variabile tablou este, de asemenea, adresa elementului zero al tabloului ⇒ după efectuarea atriburii de mai sus, *pa* şi *a* au valori identice.

♦ Mai mult, dacă pa = a, *pa+i* este adresa lui a[i]

**(pa+i)* este continutul lui a[i]

⇒ **(pa+i)* este echivalent cu a[i]

(pa+i) este echivalent cu &a[i]

♦ Când un nume de tablou este transmis, ca argument, unei funcții, ceea ce se transmite este adresa locației elementului inițial. Aceasta înseamnă că parametrul formal este de fapt un pointer, adică acționează ca o variabilă ce conține o adresă:

Exemplu: *int f(char s[ ]) { …. }*

*int f(char *s) { …. }*

/* cele două forme ale parametrului formal sunt echivalente */

- **Aritmetica pointerilor:** Spre deosebire de alte limbaje, C are facilităţi foarte puternice în acest sens.

Operaţii permise:

- atribuirea pointerilor de acelaşi tip;
- adunarea sau scăderea unui pointer cu un întreg;
- scăderea sau compararea a doi pointeri care indică spre elementele aceluiaşi tablou;
- atribuirea valorii zero (NULL) sau compararea cu aceasta.

Operaţii ilegale:

- adunarea a doi pointeri;
- înmulțirea sau împărțirea pointerilor;
- deplasarea pointerilor sau aplicarea unei măşti;
- adunarea pointerilor cu valori reale.

- **Limbajul de programare C permite definirea de pointeri la funcții** care pot fi atribuiți, plasați în tablouri, transmişi ca parametri funcțiilor, returnați ca rezultat al unor funcții etc.

În cap. 4 (§4.5) s-a arătat cum se poate folosi această facilitate pentru a transmite funcții, ca argumente altor funcții.

- **Alocarea şi relocarea dinamică a memoriei**

Ca şi în Pascal şi limbajul C permite crearea prin alocare dinamică a unor obiecte anonime ce urmează să fie referite prin pointeri:

- *malloc* şi *calloc*: alocă dinamic blocuri de memorie de dimensiune precizată

- *free:* eliberează zone de memorie punctate de pointeri care au obținut valoare prin *malloc* şi *calloc*.

Eliberarea de memorie poate să conducă la referințe fictive.

5. Structuri recursive

Pointerii permit definirea de structuri recursive (liste, arbori):

```
struct arbore {
    tip informatie ;
    struct arbore *fiustang ;
    struct arbore *fiudrept ;
}
```

✓ *Observație:* Recursivitatea, în acest caz, se poate realiza doar prin pointeri pentru că **este ilegal ca o structură să conțină propria sa instanțiere.**

6. Echivalența tipurilor

- În limbajul C se aplică *echivalența structurală*.
- Excepții: definițiile **struct** şi **union** se consideră tipuri diferite chiar dacă au structură identică.
- C permite conversii de tip explicite prin casting:

(nume_tip) expresie